NetLogger Quick Start Guide

version 3.2, 1-15-2005

Brian Tierney
Contact: bltierney@lbl.gov
http://dsd.lbl.gov/NetLogger/

Table of Contents

1 Overview of NetLogger

1.1 What is NetLogger?

Anyone who has ever tried to debug or do performance analysis of complex distributed applications knows that it can be a very difficult task. Problems and bottleneck may be in many various software components, hardware components, networks, operating systems, and so on.

NetLogger is designed to make debugging and performance analysis of complex distributed applications easier. NetLogger is both a methodology for analyzing distributed systems, and a set of tools to help implement the methodology. In fact, you can use the NetLogger methodology without using any of the NetLogger tools.

The key idea behind the NetLogger analysis method is to generate "lifelines" that represent the workflow of a distrubuted process. Lifeline analysis is very intuitive: the slope of the line indicates where the most time is being spent. The figure below contains an example of this.

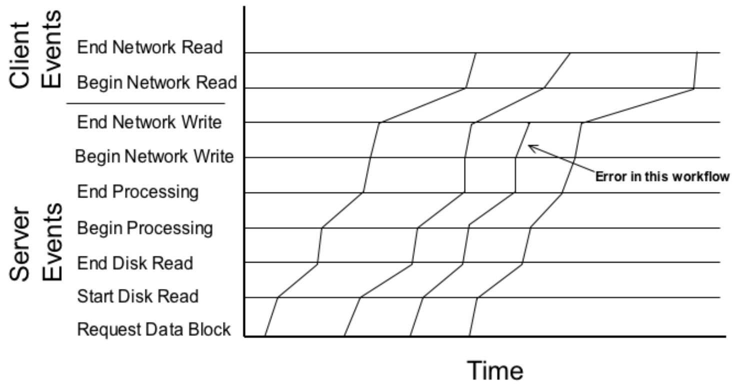

Figure 1.1: NetLogger Workflow Lifelines

This purpose of this guide is to get you using the NetLogger Toolkit as quickly as possible. More details are available in the user manual.

1.2 NetLogger Toolkit features and benefits

The Netlogger Toolkit includes a number of separate components which are designed to help you do distributed debugging and performance analysis. You can use any or all of these components, depending on your needs.

These include:

- NetLogger message format and data model: A simple, common message format for all monitoring events which includes high-precision timestamps
- NetLogger client API library: C/C++, Java, PERL, and Python calls that you add to your existing source code to generate monitoring events. The destination and logging

level of NetLogger messages are all easily controlled using an environment variable. These libraries are designed to be as lightweight as possible, and to never block or adversely affect application performance.

- NetLogger visualization tool (nlv): a powerful, customizable X-Windows tool for viewing and analysis of event logs based on time correlated and/or object correlated events.
- NetLogger host/network monitoring tools: a collection of NetLogger-instrumented host monitoring tools, including tools to interoperate with Ganglia and MonALisa.
- NetLogger storage and retrieval tools, including a daemon that collects NetLogger events from several places at a single, central host; a forwarding daemon to forward all netlogger files in a specified directory to a given location; and a NetLogger event archive based on mySQL.

Sample NetLogger Analysis

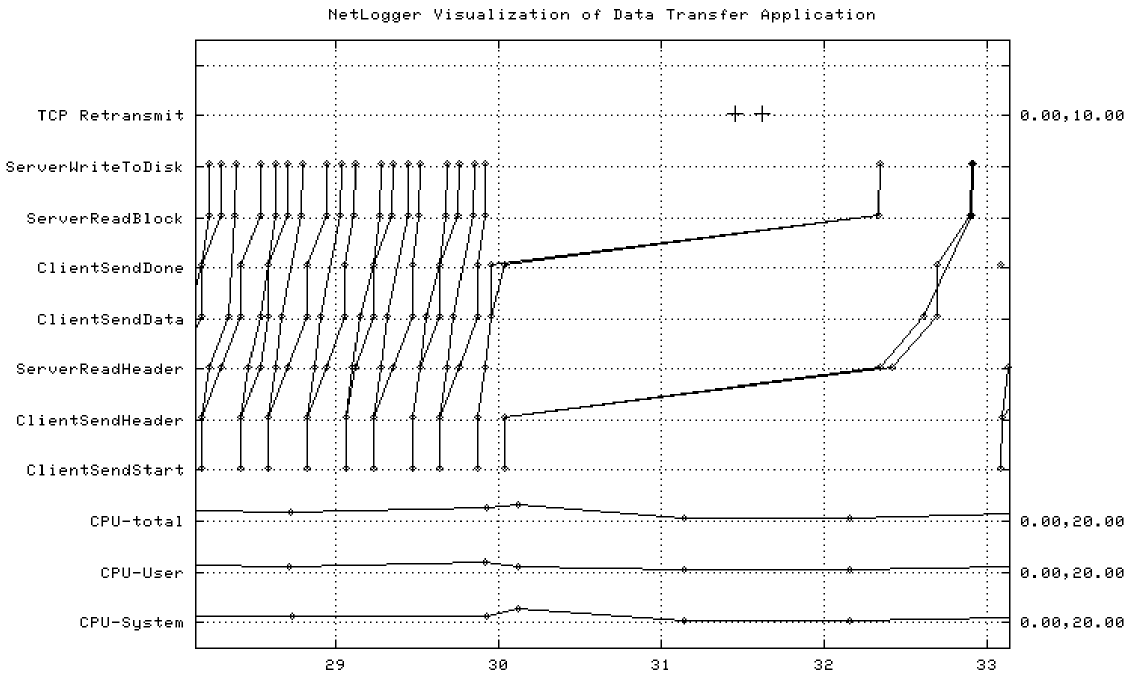

Figure 1.2: Sample NetLogger analysis

The following is an example of NetLogger's powerfull abilitiy to correlate the application and middleware instrumentation data with the host and network monitoring data. The figure above shows the result of such an analysis. This plot, generated by the NetLogger visualization tool, *nlv*, correlates client and server instrumentation data with CPU and TCP retransmission monitoring data. The events being monitored are shown on the y-axis, and time is on the x-axis. From bottom to top, one can see CPU utilization events, application events, and TCP retransmit events all on the same graph. Each semi-vertical line represents the *life* of one block of data as it moves through the application. The gap in the middle of the graph, where only one set of header and data blocks are transferred in three seconds, correlates exactly with a set of TCP retransmit events. The plot makes it easy to see that the pause in the transfer is due to TCP retransmission errors on the network. Thus, we determine that our performance problems are due to network congestion. Tracking down the cause of the network congestion is a difficult problem, and requires a large amount of network monitoring data.

1.3 Getting more Information

- **User manual**

  An extensive user manual is provided detailing the use of NetLogger
- **FAQ**

  Several Frequently Asked Questions are outlined in the NetLogger FAQ.
- **Tutorial**

  Over a hundred tutorial viewgraphs covering every aspect of NetLogger set up and use.
- **Publications**

  Several conference papers have been published on NetLogger
- **E-mail list** (coming soon)

  Send questions on any NetLogger subject to netLogger@portnoy.lbl.gov

  You can subscribe by going to the website:
  `http://dsd.lbl.gov/mailman/listinfo/NetLogger/`,
  or by placing the following command in either the subject or the body of a message addressed to NetLogger-request@portnoy.lbl.gov.

  ```
  subscribe [password] [digest-option] [address=<address>]
  ```
- **Website**

  The official NetLogger website is located at:

  `http://dsd.lbl.gov/NetLogger/`.

  It contains all of the above documentation and more.

2 Requirements

NetLogger tools should run on any Unix platform. They have been tested on Linux, Solaris, FreeBSD, and Mac OSX. The NetLogger Visualization tool nlv currently only works with Linux.

Full installation of all the NetLogger Tools depends on the following tools being installed.

- NTP (`http://www.ntp.org`), for clock synchronization
- Python version 2.3 or higher (`http://www.python.org`)
- Perl version 5 or higher (`http://www.perl.org`), for PERL client API only
- mySQL (`http://www.mysql.org`), for the NetLogger archive only

3 Installation

3.1 Download

Download NetLogger from: `http://dsd.lbl.gov/download.html`

We recommend you start with the latest "stable release". nlv is a separate download, and is distibuted in binary format only. nlv is available at `http://dsd.lbl.gov/NetLogger/nlv/Download.html`.

3.2 Install

Installation is very simple. First you set an environment variable with the directory to install NetLogger into. For example:

```
setenv NLHOME /usr/local/NetLogger
```

Then type the following:

```
make unpack
make
make install
```

For Redhat Linux an alternate method is to download the NetLogger RPM and install that.

To use the NetLogger tools, add NetLogger to your binary and library search paths. For example:

```
add $NLHOME/Library/Perl/ to PERL5LIB
add $NLHOME/lib/python<ver>/site-packages to PYTHONPATH
add $NLHOME/lib/java/*.jar to CLASSPATH
add $NLHOME/bin to PATH
```

Time Synchronization

To analyze NetLogger events from multiple hosts requires that the clocks of all hosts be synchronized as well as possible (e.g.: within about 10 ms). You can use NTP for time synchronization. NTP is included in most Unix distributions. If you do not have a local time server to synchronize with, you can use one of the public time servers.

4 Using NetLogger

This chapter provides a short overview of how to instrument your application with NetLogger, and then collect and analyze the results.

4.1 Instrumenting Applications with NetLogger

The first step is to instrument your applications using the NetLogger Toolkits's client library. If you have access to the source code, insert NetLogger calls into the applications at all key points. If you don't have the source code, you can write a wrapper script in python or perl which generates netlogger messages just before and just after the program runs.

In general you should add instrumentation to the following locations.

- entering and leaving the program
- before and after all I/O (network and disk)
- before and after any compute intensive operation (e.g.: a large matrix multiplication)

The basic method for all language bindings of the client API is that you first create a NetLogger *logger module*, and then use one of the NetLogger *write* commands to generate and output a NetLogger *event*. Details for the sytax for doing this in C, Java, Python, and Perl are in the User Manual.

Here is a simple example in C:

```
#include "nl_log.h"
main() {
  int i;
  NL_logger_module("myapp",NULL, NL_LVL_DEBUG, NL_TYPE_DBG, "");
  for ( i=0; i < 10; i++ ) {
        NL_write(INFO, "myapp","doit.start","index=i", i);
        do_something();  /* monitoring this */
        NL_write(INFO, "myapp","doit.end","index=i", i);
  }
}
```

For Java and Python we provide NetLogger provides APIs that are compatible with the Python logging module and the Java log4j API.

For example, to use the Java API, you need a *properties* file that contains something like this:

```
     # Layout for NETLOGGER appender
     log4j.appender.NETLOGGER.layout=org.apache.log4j.PatternLayout
     log4j.appender.NETLOGGER.layout.ConversionPattern=s LVL: %p%n%m
```

See the User Manual for more details.

NetLogger Message Format

The NetLogger Instrumentation APIs generate NetLogger *events*. The event is subdivided into fields. Each field has three parts:

```
     typecode key: value
```

NetLogger supports the following typecodes:

```
d Double. Double-precision floating-point number
i Integer. Signed 32-bit integer
l Long (64-bit) integer
s String
t Timestamp. YYYY-MM-DDThh:mm:ss.ssssss (ISO8601 date format)
```

Here is a sample NetLogger event:

```
t DATE: 2004-04-15T21:36:01.425059
s LVL: Info
s HOST: foo.lbl.gov
s TGT: app
S PRGM: test_program
s EVNT: SendData
i Send.Size: 49332
```

This says program named test_program on host foo.lbl.gov performed event named SendData, size = 49332 bytes, at the time given.

Naming Conventions

We recommend that you use descriptive, hierarchical event names. For example, see the Global Grid Forum's Discovery and Monitoring Event Description working group's event names. Dots or underscores can separate parts of event names.

Some sample event names might include:

- myapp.start, myapp.end
- myapp.disk.read.start, myapp.disk.read.end
- myapp.network.write.start, myapp.network.write.start
- myapp.db.query.start, myapp.db.query.end

Adding NetLogger events to a distributed process is usually an iterative processes. Often one will gradually add more and more events until the bottlenecks are identified.

NetLogger "Lifelines"

In order to associate a group of events into a *lifeline* you must assign an *event ID* to each NetLogger event. An event ID can be any unique identifier that can be used to group events together. Some sample event IDs include:

- input file name
- data block ID
- image/video frame ID
- batch system job ID

The event ID is then passed to the NetLogger *write* call as a user-defined field. For example, here is C code to generate NetLogger events with the *data block index* used as the event ID.

```
#include "nl_log.h"
main() {
  int i=0, size = 65536, num_blocks = 100;
        NL_logger_module("foo",NULL, NL_LVL_INFO, NL_TYPE_APP, "");
  for ( i=0; i < num_blocks; i++ ) {
  /* read block of data */
        NL_write(INFO, "foo","read.start","index=i read.size=i", i, size);
        read_data(i);
        NL_write(INFO, "foo","read.end","index=i read.size=i", i, size);
   /* process block of data */
        NL_write(INFO, "foo","process.start","index=i",i,);
        process_data(i);
        NL_write(INFO, "foo","read.start","index=i read.size=i", i, size);
   /* write results */
        NL_write(INFO, "foo","write.start","index=i write.size=i", i, size);
        send_data(i);
        NL_write(INFO, "foo","write.start","index=i write.size=i", i, size);
  }
}
```

NetLogger Output Destination

NetLogger can write it's log files to local disk, or to the NetLogger collection daemon, netlogd.

The destination is specifed using URL syntax, and passed into NetLogger when creating a NetLogger module, or set using the NETLOGGER_DEST environment variable.

Here are some sample NetLogger URLs:

file://tmp/netlogger.out	write to file /tmp/netlogger.out
/tmp/netlogger.out	write to file /tmp/netlogger.out
x-netlog://myloghost.lbl.gov:7234	write to netlogd on host myloghost, port 7234

Debug levels

NetLogger supports up to 255 different logging levels, with level 0 the default. This can be very convenient for debugging. For example, you might want the default NetLogger level to just log program.start and program.end. Level 1 might add IO monitoring. Level 2 might add more application monitoring, and level 3 might be very detailed debugging information.

Environment Variables

NetLogger uses the following environment variables.

NETLOGGER_DEST	Sets destination URL for NetLogger data
NETLOGGER_ON	Turn NetLogger on/off by setting this to 0/1
NLCONFIGLEVEL	Filename for file containing default log level
NL_GID	Use to pass in a "Grid ID". If this is set, NetLogger adds GID=$NL_GID to all events, which can be used to group events together into "lifelines".

4.2 Monitoring System Information with NetLogger

NetLogger also includes the following host monitoring tools. These can be used to collect host monitoring data to correlate with your application instrumentation data.

- nlcpu.py: Linux CPU sensor
- nlganglia.py: Ganglia wrapper
- nlmem.py: Linux memory usage sensor

4.3 Activating NetLogger

To change the NetLogger debug level that gets output, set the environment variable *NL-CONFIGLEVEL* to point to a file containing the default log level. You can change this *on-the-fly* in a running program by doing this:

```
setenv NLCONFIGLEVEL=/tmp/nl_level
```

Then start your program, and if you decide you need to increase the Level of instrumentation, you can then do this:

```
echo 3 > /tmp/nl_level
```

The default logging level will be changed on the fly. This file is checked every 10 seconds.

4.4 Collecting Results

NetLogger can write it's log files to local disk, or to the NetLogger collection daemon, netlogd. Depending on the amount of logging data being generated, it may be safer to write the logs to local disk space, and then use nlforward to forward the results to `netlogd`.

For example, to forward all files in '`/tmp/netlogger`', re-scanning the directory for new files every 10 minutes:

```
nlforward.py -s 600 /tmp/netlogger x-netlog://remote.host
```

4.5 Automatic Instrumentation Support

NetLogger includes support for automatic instrumentation for Python programs.

At run-time, NetLogger can automatically instrument a Python module, class, or (list of) functions. A NetLogger write call is inserted before and after the function invocation. The logging object, function called to write a message, event name suffixes, etc. are all configurable. For an example see autolog.py.

4.6 Data Analysis

Using gnuplot

The NetLogger tool nldata can be used to extract data from NetLogger log files. The fields to extract are specified in a config file. Then nldata_gp can be used to format the file for use with gnuplot.

For example, to display a gnuplot on the terminal from 'my.log', configuring how to graph it from 'my.cfg'.

```
cat my.log | nldata.py my.cfg | nldata_gp.py | gnuplot -persist
```

Using the NetLogger Visualization Tool (nlv)

Often exploratory, interactive analysis of the log data is an invaluable means of identifying problems. This is provided by *nlv*, which has the following features

- ability to display several types of NetLogger events at once
- user configurable: which events to plot, and the type of plot to draw (lifeline, load-line, or point)
- play, pause, rewind, slow motion, zoom in/out, and so on
- nlv can be run post-mortem or in real-time. Real-time mode is performed by reading the output of netlogd as it is being written

To use nlv you must first generate a configuration file defining how nlv should plot your log data, based on the event names that you have specified. Documentation on creating nlv configuration files is in the nlv manual.

Index